TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Vijeth Shetty

Commanding an Indoor Robot with Gaze Using Eye Tracking Glasses and Motion Capture Camera

Master of Science Thesis

Examiners:
Prof. Risto Ritala
Prof. Pasi Kallio

Examiners and thesis topic approved by Faculty Council of Faculty of Engineering Sciences on 26th April 2017

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Engineering
Shetty,Vijeth: Commanding an Indoor Robot with Gaze Using Eye Tracking Glasses and Motion Capture Camera.
Master of Science Thesis, 50 pages, 11 Appendix pages
August 2017
Major: Mechatronics and Micromachines
Examiners: Prof. Risto Ritala, Prof. Pasi Kallio
Keywords: Gaze Commanding, TurtleBot, Motion Capture ,Eye Tracking Glasses

For a robot to navigate any environment, it should have a perception of its position. This is achieved by robots ability to create a virtual map of the environment or by its ability to keep track of its position using global positioning system. With the availability of motion tracking systems, it is possible to have an independent system which can keep track of the robot.

This thesis tries to use Eye tracking glasses in conjunction with Motion tracking system to command and guide a TurtleBot around a predefined 3D space. This is achieved with the help of Robotic system toolbox present in MatLab which helps in communicating with Robot Operating System (ROS) installed in TurtleBot, Eye tracking glasses use Virtual reality peripheral network to connect with Matlab and Motive.

Matlab scripts are used to retrieve data of gaze direction from Eye tracking glasses and calculate the final position desired by the user. The distance and orientation of final position is calculated with respect to the present position of the Robot. It is oriented and moved towards the final position while being tracked using motion tracking system.

This thesis managed to establish a system for communication between Eye tracking glasses, Motion tracking system and Turtlebot. Which is reliable and easy to setup.

PREFACE

This work has been carried out at the Department of Automation Engineering, Tampere University of Technology, Finland.

I would like to express my utmost gratitude to my supervisor Prof. Risto Ritala, for his constant support, invaluable guidance and immense patience throughout this work. I would also like to thank Sindhu Shetty, Gaurav Naithani, Tuomas Vlimki, Aino Ropponen and Eero Heinnen for providing me valuable suggestions which helped me learn a lot throughout this work.

Again I would like to express my appreciation to Prof.Risto Ritala for providing me with the Project idea and the department of automation engineering for providing all the tools and a friendly environment to work which made this project possible.

Finally I would like to thank people who were part of my life throughout this project and beyond .Without their sheer support, endless love and sacrifices, I would not be where I am today.

Vijeth Shetty

Tampere, 30th July 2017

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Motion tracking systems have been in the entertainment industry for the longest period as they form the backbone of creating any animations or Computer-generated imagery (CGI) sequence present in the movies. Due to the technological advancements, the cost of the system has decreased enabling educational institutions to get their hands on the system and implement them to conduct various research.

These systems are useful when automating robot systems as it is important for the robots to know their position. If the internal mapping of the robot is not well established or prone to errors it is difficult for the robot to function without interfering with the surrounding environment. Since motion tracking system is an independent system which is fixed to the area of interest and the processing of information can be carried out independently with limited communication with the robot, this helps in accurate mapping of the environment hence optimize the functioning of the autonomous robot.

Virtual Reality (VR) systems use the ability to track the eye position of the user to provide a better experience for the user. It can be adjusting the projected environment to user perspective or even improving functionality using these techniques. An example for application of eye tracking system to improve usability is the use of voice activation to select objects in the user environment [1] while using hololens, Figure 1.1.

Figure 1.1 *Example of an eye tracking systems*

1.1 Objective

The objectives of this thesis are as follows

1. To record the gaze direction using Eye Tracking Glasses (ETG) and to calculate the final position pointed by the user.
2. To control the motion of a TurtleBot (TB) to the final position with continuous tracking using motion capture system.

1.2 Contribution

This project meets the objectives by

1. studying TB, ETG and motion capture system hardware and software packages.
2. forming a system that connects all of the above and establishes control and communication between them.
3. assessing the accuracy of the robot control.

1.3 Thesis Structure

This thesis is divided into five Chapters.Chapter two familiarizes the user with robot kits, motion tracking, and eye tracking system. Chapter three explains in more detail the hardware and software in this application. Chapter four explains how the robot, the motion tracking system, and the eye tracking system were integrated to reach the objective. Chapter five explains the results of the project and how the errors were minimized. The last chapter summarizes the whole project with suggestions on future work.

2. BACKGROUND

2.1 Robot kits

A robot kit is a commercially available kit containing robot components such as motors, sensors, electric wires and various frames for assembly. Robot kits can be bought by educational institutions and hobbyists alike. The user can choose which sensors and actuators to include in the robot. The body of the robot is made of a rigid metal frame, but some robot kits for example are made of Lego bricks.

Most of these kits run on open source software like Robot Operating System (ROS) which has its own community of users. This makes it easier in sharing robot design and programs. Sharing helps newcomers to the field in taking an interest in robotics and learn from community projects by applying the acquired knowledge in practice. Some of the readily available robot kits like Lego Mindstorm, Arduino robot kit and TurtleBot are discussed below.

Lego Mindstorms is one of the publicly available robot kits which is designed for users with varied ages. As the name suggests, the kit comes with a programmable controller called a Robotic Command eXplorers(RCX) brick, around which the Lego is assembled to form the body of the robot connected to sensors and actuators. Programmable brick comes with a standard programming language called NXT-G, which is newbie friendly. Advance users can program it with Matlab, Labview, C++, Python and other languages [2]. These kits can be used to build advanced robots, for example Figure 2.1 shows a customized robot built to carry out test tube filling with precision [3].

Arduino robot kit can be bought off the market for an affordable price. The robot kit comes with an Arduino controller, sensors and motor wheels. These controllers are easy to program and deploy. This Plug and Play process decreases the time spent on technical aspects. These Arduino controllers are very inexpensive and versatile, which works in favor of the user to try out robotic applications with little effort [4]. They are supported with Arduino's own Programming language which is similar in syntax to other programming languages but are stripped down to make them

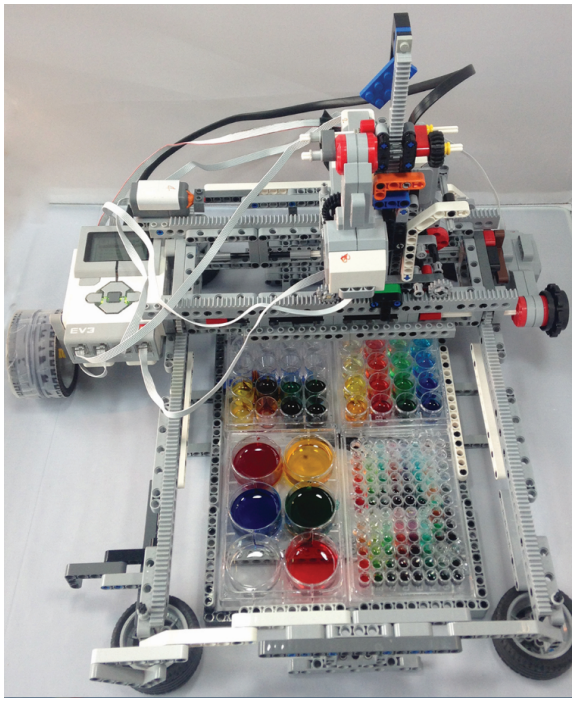


Figure 2.1 *Test tube filling system build using Lego Mindstorms*

suitable for beginners [5]. Figure 2.2 shows an Arduino robot kit with actuator and Arduino controller.

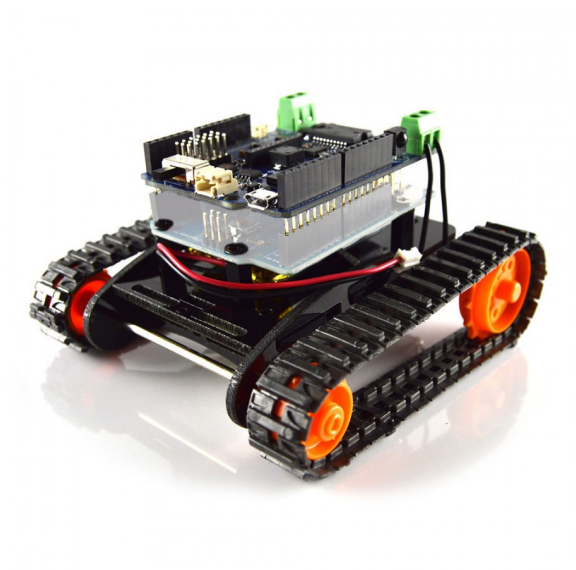


Figure 2.2 *A simple robot build using Arduino robot kit*

Turtlebot is a Robot kit which is designed mainly for educational purposes. These kits are of higher price range and come with features which aim at broader application related to research and education. Turtlebot will be discussed in detail in Section 3.1 as it is used in this project and hence requires a more detailed overview

to understand it.

2.2 Motion tracking

Motion tracking is the process of tracking the position and movement of an object in a fixed or open environment. This is accomplished with the help of special cameras designed for this purpose. There are two main methods of motion tracking: video tracking and motion capture.

In video tracking, the object is recorded via video recording and each frame of the recording is compared to track the object's movement .The object is recognized using target representation and localization algorithms which requires a considerable amount of data processing. One of the major drawbacks of this method is the time taken to process the data collected and the rate at which the data is collected, i.e. frames per second(fps) of the camera which determines the maximum speed of the object that can be tracked. This method is used in day to day applications like security cameras, traffic control and video editing where cost of the system is considered over quality of motion tracking [6]. An example of video tracking system is Xbox 360 Kinect shown in Figure 2.3.

Figure 2.3 *XBOX kinect: a commercially available video tracking system*

Motion tracking is generally used in movie industry to animate objects in the form of motion capture (MOCA) system, where the movement of characters are based on the movements of an actor. The object is tracked using markers which are attached to the object and the reflected light from the object is detected. Markerless MOCA tracks objects with machine vision which eliminate the use of external peripherals to accomplish the task [7].

Advantages of motion capture method over video tracking include faster processing time and its not restricted by the amount of data collected during longer run time making it cost effective. However, it requires high initial investment for hardware, software licensing and it also requires a fixed space where the cameras can be setup and used.

Figure 2.4 *An active marker based motion capturing system used for Playstation console.*

In marker based motion tracking system, the markers attached to the body of the tracked object can be of two types, active and passive. An active marker illuminates itself with the help of LED which is connected to a battery whereas a passive marker is a surface which can reflect the light generated externally to the camera surface. Active markers have low noise resulting in better measurements and its possible to differentiate between objects by using different colors of LED. An example is shown in Figure 2.4. On the other hand, when using passive markers there is no need for wires and batteries to power the marker.

Motion tracking cameras in the market are categorized on the basis of frame rate, resolution and field of view. Resolution determines the quality of image recorded by the camera, a high resolution camera helps in obtaining better recording which helps in better data analysis. Frame rate helps in determining how well the motion can be tracked for faster objects and field of view helps in calculating how many cameras are required to capture a given volume.

Motion tracking cameras are connected to a computer with the software supplied by the vendor. The software receives data from all the cameras and projects it on a virtual three dimensional (3d) space which is setup during calibration. Cameras need to be calibrated to define the 3d space and the origin of the coordinate system. This also improves tracking as it can avoid noise generated around the area of interest, for example undesired reflection of light present in the capture area .

The user can control the frames per second(fps), exposure and threshold of the cameras in the software which helps to tune the image quality depending on the environment. Fps defines how many frames are captured by the camera group per second. Exposure defines how long the camera exposes per frame, hence lower fps allows higher exposure time. Threshold defines the minimum illumination of the

marker required to be registered by the camera. These options help in fine tuning the tracking of the object. For example, a low fps setting introduces high exposure which blurs the recorded marker leading to false marker reading [8].

When an object is defined in the 3D space, the axis is defined by X,Y and Z coordinates. Rotation values are recorded in quaternions,and as well as yaw,pitch and yaw.Due to complex visualization of rotation in quaternions, Euler angle is used, it is important to know the order in which rotation is applied. Figure 2.5 shows the Euler angle convention used in Motive. The rotation order is XYZ where pitch is rotation across X-axis,roll is rotation across Z-axis and yaw is rotation across Y-axis.

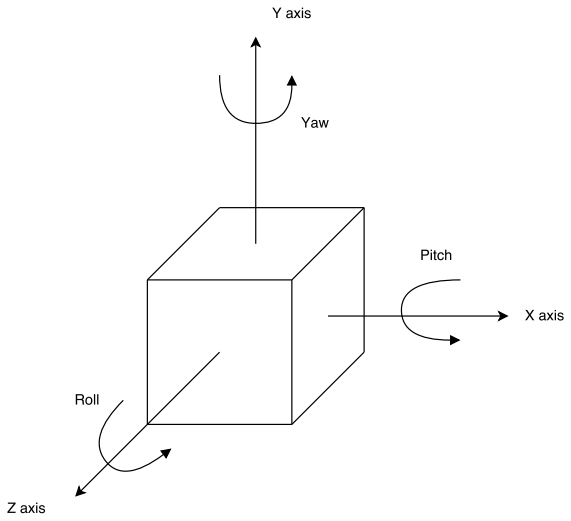


Figure 2.5 *Orientation in 3D according to Euler angle convention*

2.3 Eye tracking

Eye tracking is the process of determining the position of the eye ball with respect to head position. This information is utilized to determine the direction of gaze of the user .

Eye tracking methods are broadly categorized into contact and non contact methods. Contact methods include special contact lens which are worn by the user and the position of this lens help in tracking the position of the eye. Another type of contact method involves measuring the potential field developed by the eye movement. This is measured using small electrodes which are placed on the skin around the eye . The movement in cornea acts as a dipole moment which creates potential difference this can be measured to determine the position of eye [9].

In non contact method, infrared light is used to track the eye position. An infrared emitter illuminates the eye, whereas the receiver records the reflected light. The amount of reflected light and the change in it helps in determining the position and movement of the eye. This method can be used even in darkness with no visible light. Another type of non contact method uses a video camera in conjunction with infrared light as shown in Figure 2.6. In this method both high quality images from camera and data recorded from infrared light is used to determine the position of the eye [10].

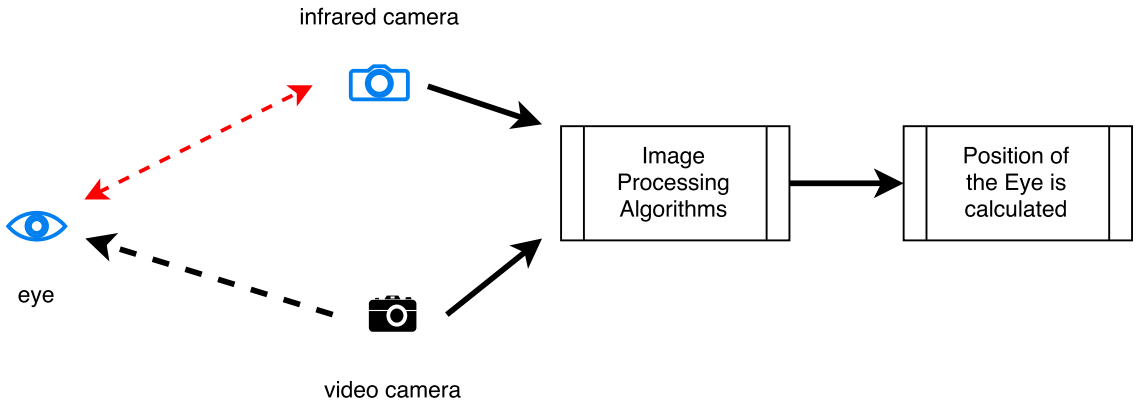


Figure 2.6 *Non contact eye tracking method using infrared and video camera*

Eye tracking is applied in neuroscience, psychology, industrial engineering, advertising and computer science [11].

3. RESEARCH INFRASTRUCTURE

3.1 TurtleBot

TurtleBot2 (TB) is the second revision of TurtleBot released in 2012. This revision replaces the netbook present in the TB with a single board computer which drastically reduces the price and power requirements, and this increases the runtime of the Robot. The robot kit includes a Kobuki mobile robot base, laptop, kinect modification kit, docking station and frame of the TB [12].

The robot is assembled on top of a Kobuki mobile robot base, see Figure 3.1, which is powered by a 4400mAh Li-ion battery. It powers the sensors and actuators connected to the TB. The laptop is connected to the base and powered by a separate battery. Robot Operating System (ROS) is installed on the laptop which connects and controls the actuator wheel and odometer, gyroscope sensor on the robot base. The robot base can reach the speed of 70 centimeters per second and a rotational velocity of 180 degrees per second. TB also has three bumpers positioned in the front and sides to detect any collision which can be used individually in user programs [13].

ROS is the backbone of communication for TB. It provides the user with libraries and tools to create applications. ROS is made of packages which collectively constitute a stack. A stack runs the system where as the packages in the stack run individual sensors. Hence if we were to add a new sensor then the package is modified and added to the stack.

Figure 3.1 *TB assembly process with Kinect installation on Kobuki base*

A package consists of a node, a node contains a set of instructions to carry out a task. In order to carry out a large task, it is advisable to write a set of smaller nodes rather than a large single one to keep the system modular. The communication between nodes is carried out via Messages using Topics.

Messages are a set of data structures consisting of typed fields, such as integer, floating point, boolean and arrays. During communication, a ROS master helps in connecting and finding all the nodes which are registered to it. For example, Node 1, Node 2 and Node 3 are registered to ROS master. Once Node 1 receives data from sensor, it publishes the data to the Topic. The Client(laptop) will then receive the data at Node 3 as shown in Figure 3.2.

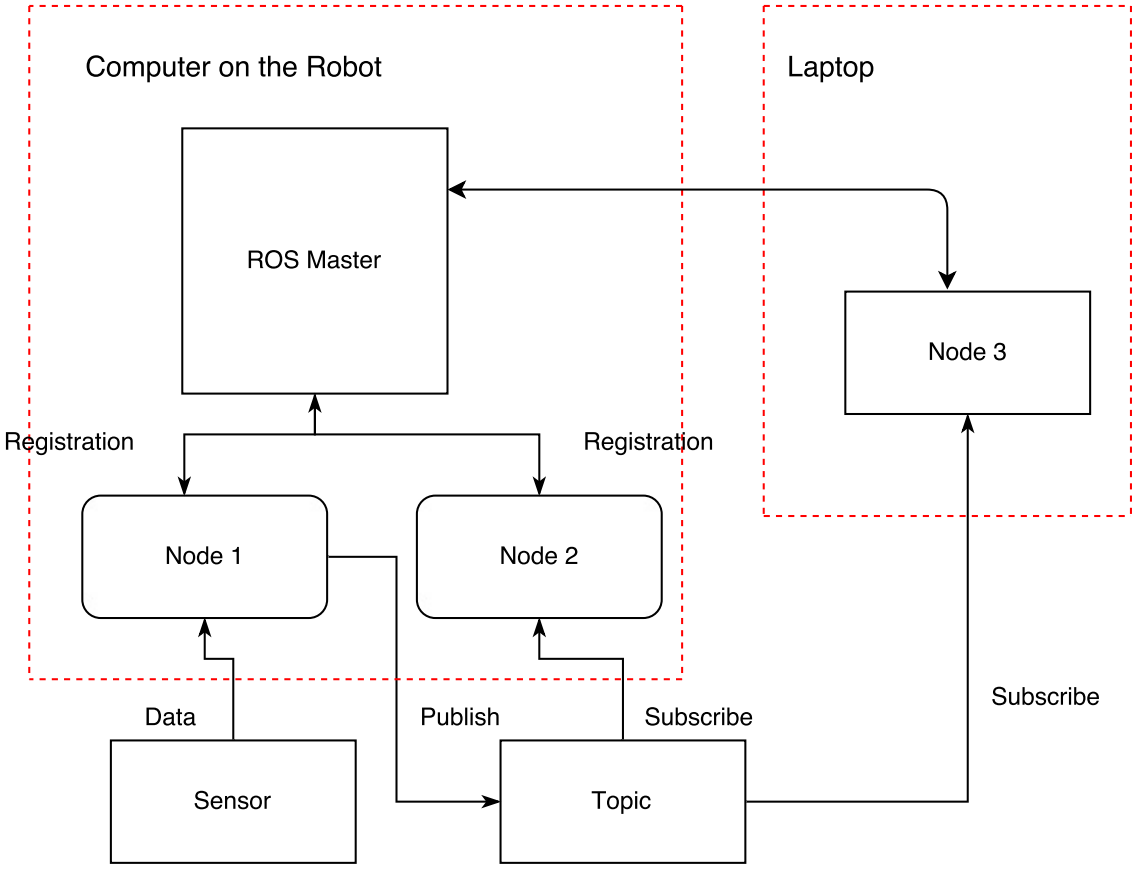


Figure 3.2 *Communication in ROS system*

Robotics System Toolbox is an interface between Matlab, Simulink and ROS providing tools for communication: this provides opportunities for interaction with the robot, collection and visualization of sensor data on MATLAB .With this , the user can create and develop algorithms on Matlab and Simulink but communicate with the robot using nodes via ROS network .The way nodes works in this environment is similar to the one mentioned previously.

To initiate Matlab ROS system, a running ROS master is required. A valid IP

address is required to initiate the subsystem using the command

```
rosinit('192.168.2.59');
```

In order to create a publisher which can communicate with the topic, its important to know the topic name and its message type. For topic name '/chatter' with data type 'String', publisher is created as follows

```
chatterpub = rospublisher('/chatter', 'std_msgs/String')
```

to send 'hello world ' as a message to the topic '/chatter' the following command is used. Its important to note that the message type suits the predefined message type of the topic.

```
chattermsg = rosmessage(chatterpub);
chattermsg.Data = 'hello world'
```

A subscriber can now subscribe to the topic in order to receive the message. In order to check whether the topic is currently available, 'rostopic list' command can be used.

```
chattersub = rossubscriber('/chatter')

chattersub =

 Subscriber with properties:

       TopicName: '/chatter'
     MessageType: 'std_msgs/String'
   LatestMessage: [0  1  String]
      BufferSize: 1
```

Now the Message can be published to the topic '/chatter' using 'send(chatterpub,chattermsg)', which displays the message 'hello world' which was present in 'chattermsg.Data'.

```
send(chatterpub,chattermsg)
pause(2)
Chatter Callback message data:

ans =

    'hello world'
```

After the communication is complete , the network can be shutdown at anytime using [14].

```
rosshutdown()
```

3.2 OptiTrack Motion Capture System

OptiTrack is one of the manufactures of motion capture systems, which are used in both research and entertainment. OptiTrack provides systems for functions such as Virtual Reality (VR) , Movement Sciences, Robotics and Animation. Depending on the purpose you can choose suitable cameras with different functionalities. The amount of cameras used depends on the application and capture area[15].

OptiTrack system used in this project consist of Flex 3 Cameras, the camera can record at 100fps with 640x480 resolution. It comes with 26 LED for illumination with a maximum horizontal field of view of 58 degree. The present setup is suitable for capturing one subject [16].

The cameras are placed on equal elevation with camera mounting brackets fixed on the walls. They are placed equidistant from each other so that they can efficiently capture the 3D volume. The placement of the camera is important for motion tracking as three markers should be in view of the camera at all times. The cameras can be connected directly via an USB 2.0 connection or for multiple cameras an OptiHub system is used. These Hubs can connect upto 6 cameras and are powered by 12V input. When multiple hubs are used, they need to be synchronized with a synchronization cable, camera setup is shown in Figure 3.3.

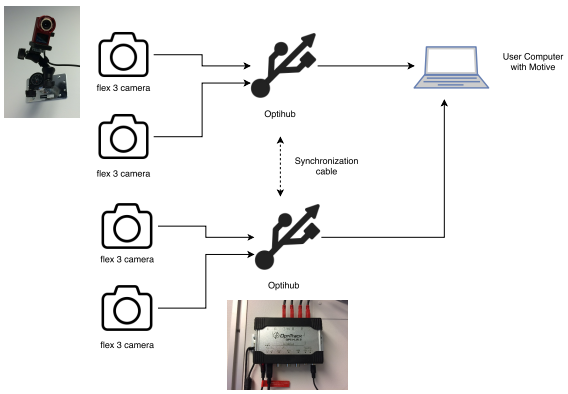


Figure 3.3 *An example setup of Optitrack motion capture system*

The cameras are controlled via software provided by OptiTrack, called Motive. This helps in calibrating and controlling the cameras, defining capture volume, managing projects, recording motion capture, live streaming data to other network and more.

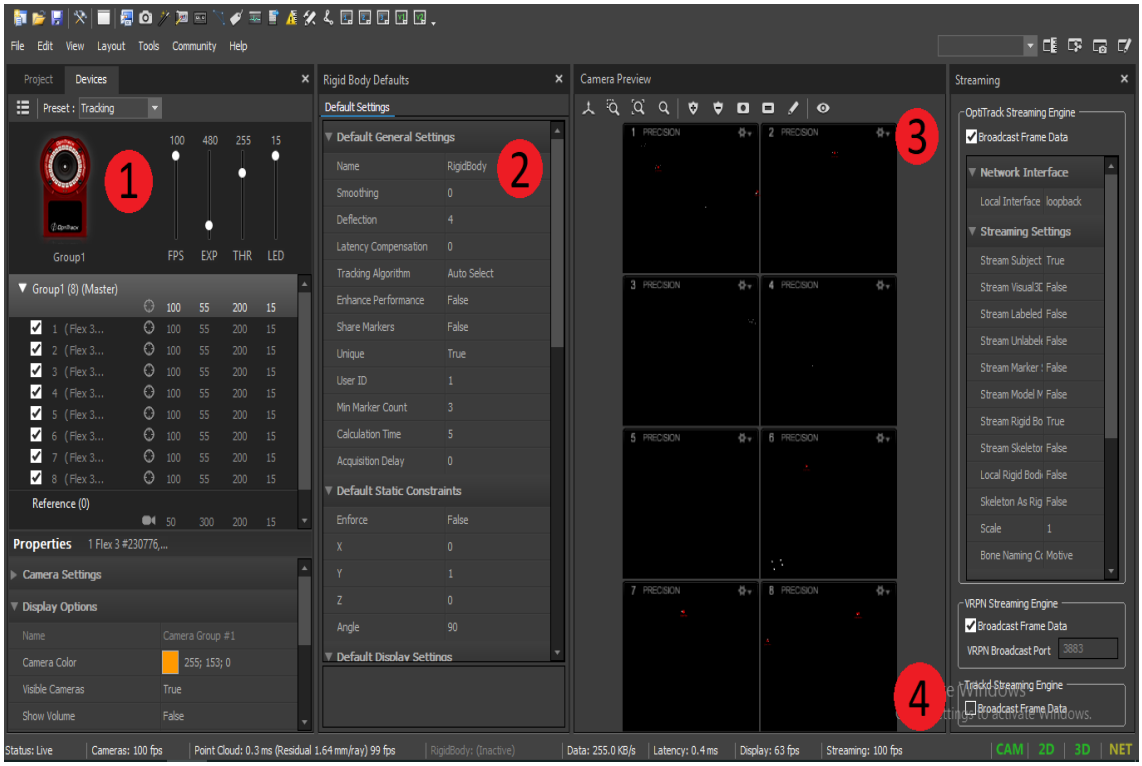


Figure 3.4 *UI of Optitrack software and software setting available for the user*

The user can preview each camera's 2D view or can choose to have a 3D perspective of combined cameras as shown in Figure 3.4. Numbers denote the components in user interface. Area 1: Show the number of cameras currently online with an option to switch off particular cameras; It also gives control to frames per second, exposure, threshold and led illumination. Area 2: helps in choosing various properties of rigid body recorded for later. The user can choose the color, pivot point, rigid body name and id and options like viewing the orientation planes on the camera preview pane. In Area 3: the user can choose either to look at individual camera viewpoints or a combined view as shown in Figure 3.5. Area 4: helps in choosing live streaming settings, such as local host address, port number for VRPN connection.

The subject with markers is placed in the capture volume in order to define assets in Motive. The subject can be defined as a marker set, rigid body or skeleton. Marker set is used to define markers which are attached to flexible object whereas rigid body defines rigid objects. Skeleton tracking is used generally for human tracking due to individual part movements.

When a rigid body is defined it is tracked using a pivot point, that defines both orientation and position of the body. The software automatically places the pivot point at the center of the body with its orientation along the global coordinate axis. This can be adjusted within the software by the user, which helps in correcting

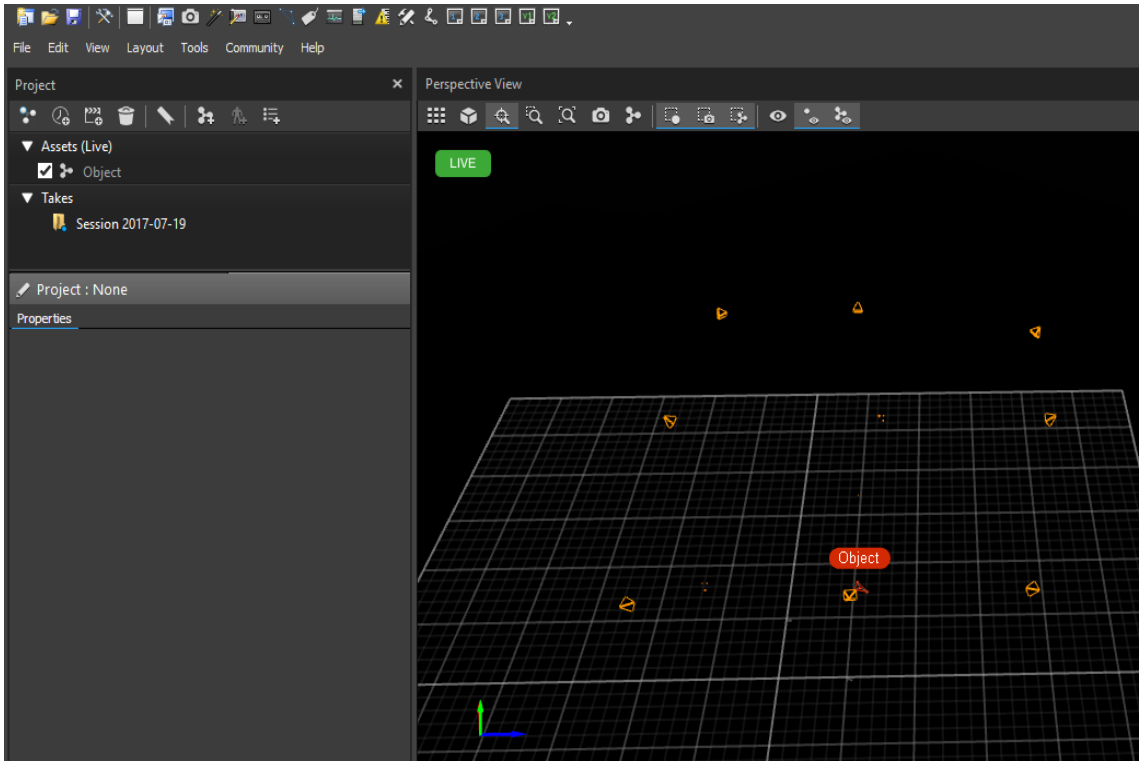


Figure 3.5 *Example of an Object defined in Motive environment*

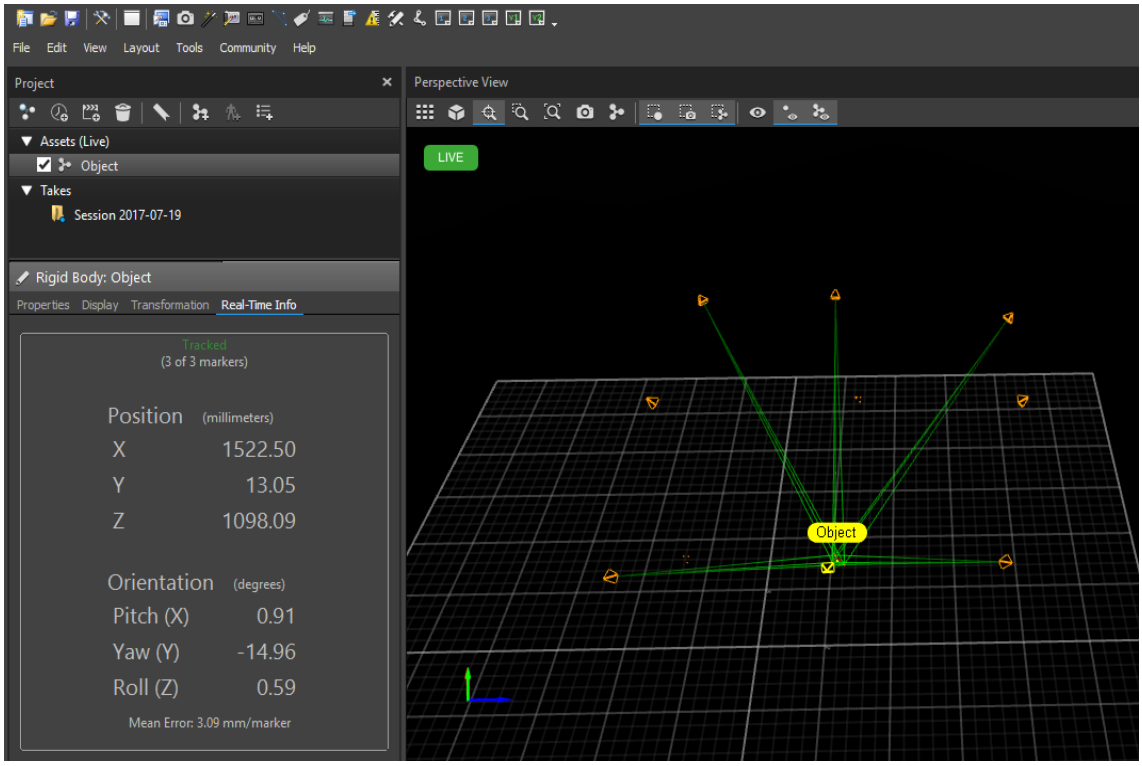


Figure 3.6 *Live tracking of an object in Motive environment, highlighted with green lines*

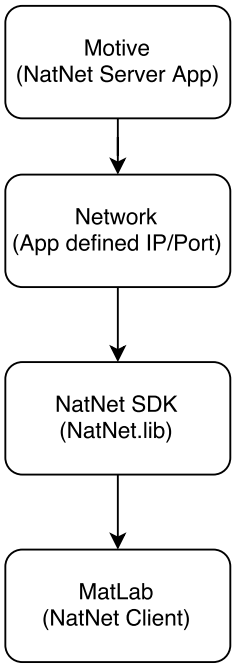


Figure 3.7 *Componenets of a NatNet network*

the pivot point when the user chooses asymmetric shapes. The rigid bodies are differentiated by the software with the help of rigid body ID. It is a number which is given to a rigid body which helps in associate streamed data to rigid bodies.

Streaming in Motive can be accomplished using NatNet client/server[17] networking protocol. This allows real time streaming of rigid body data from Motive to user applications. NatNet SDK provides a set of libraries which helps in creating applications that establish connection between MATLAB and Motive. Data can also be streamed to Virtual-Reality Peripheral Network (VRPN) [18] , an open source application, which is used to create servers for providing communications between different application and hardware used in VR system.

The user can check real time info about the position of the tracked object inside the software and this includes orientation in X, Y, Z axis measures in millimeters and Rotation across X, Y, Z axis namely Pitch, Yaw, Roll, respectively measured in degrees.This data is streamed to the local network .

3.3 SMI Eye Tracking Glasses

SensoMotoric Instruments (SMI) is a long standing manufacture of Eye Tracking Glasses (ETG). They provide eye tracking solutions for fields like clinical research, psychology, vision sciences, virtual and augmented reality. ETG uses a non contact method and comes with iViewNG software development kit (iViewNG SDK). They

Target ID	X	Y	Z
1	-75.641	32.752	17.146
2	-110.467	10.732	-9.658
3	-91.549	-49.899	-2.478
4	79.350	30.992	18.130
5	104.418	27.230	-22.249
6	93.890	-51.807	-0.891

Table 3.1 *Coordinates of markers from the origin(measurements are in mm)*

provide Application Programming Interface(API) which helps to create eye tracking applications. Currently the SDK support only C/C++ language with future updates for C sharp, python and MatLab [19].

SMI ETG consist of glasses which are worn by the user and is connected to a smart phone supplied by the manufacture or a user laptop with iViewNG Software. These glasses have a resolution of 960 pixels @ 26 frames per second with tracking range of 80 horizontal, 60 vertical with a refresh rate of 60Hz. The glasses are powered via the micro USB cable which is connected to a PDA or laptop. It uses non contact method for eye tracking, where as for head tracking the glasses are provided with a snap-on frame consisting three reflective markers on each side.

ETG with snap on frame are defined as a rigid body in the Motive environment and this data is streamed live to the VRPN server. The VRPN server files are provide by SMI which take the data from ETG and Motion tracking system and merge them and stream it to the local network on a desired port. The UI interface gives the option to specify the the IP address of ETG, motion tracker software which have predefined settings and iView 3D6D port for accessing merged data. The connections are color coded in red, yellow and green to notify no connection, connection with no valid data being transmitted, and successful connection with data transmission respectively as shown in Figure 3.8 [20].

In order to access this data the user has to create their own application which can be written as a C/C++ program. The header files are provided along with the 6D head tracking kit supplied by SMI. The flow of data in an VRPN network is shown in Figure 3.9

The relative position of the reflective markers are fixed and are shown in Table 3.1 and their respective points in the ETG are shown in Figure 3.10

The merged data from the VRPN server consists of the following

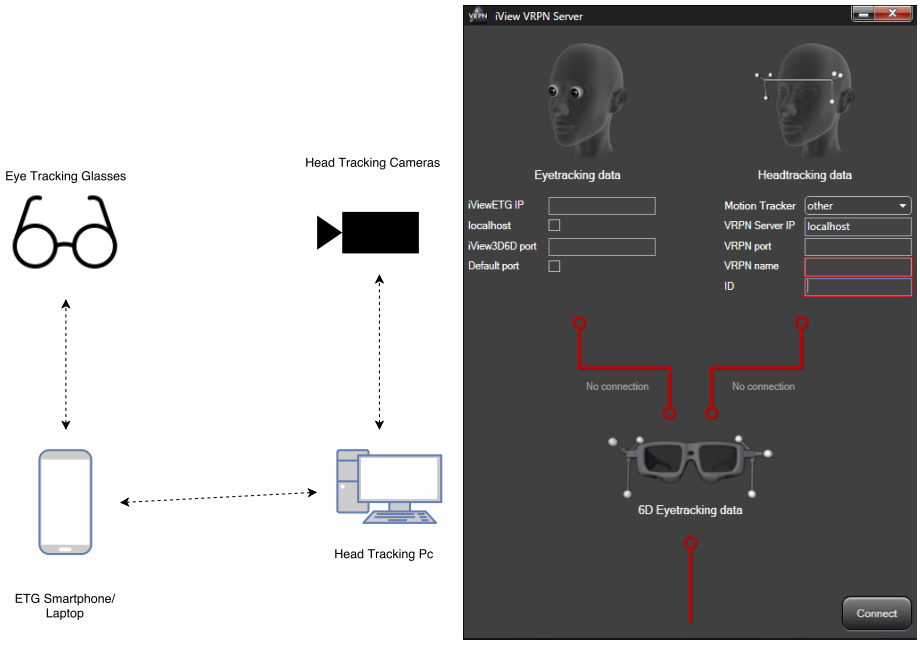


Figure 3.8 *System setup of ETG glasses (left) VRPN server UI with no active connections*

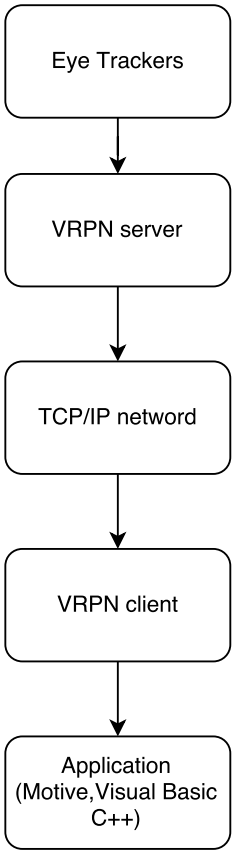


Figure 3.9 *Work flow of a VRPN server and Client to write data from application*

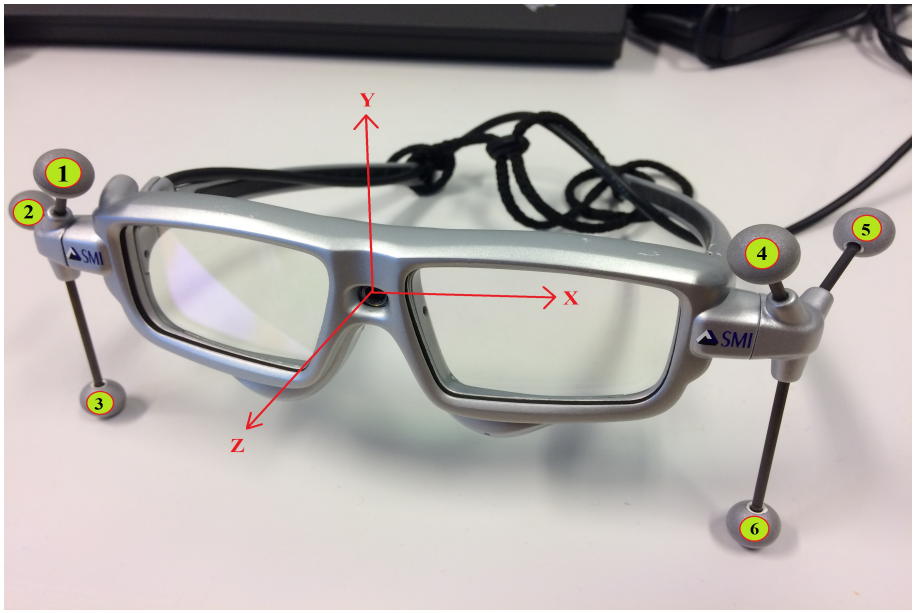


Figure 3.10 *ETG with head tracking module*

```
typedef struct _vrpn_TRACKERCB {
                struct timeval msg_time;
                vrpn_int32 sensor;
                vrpn_float64 pos[3];
                vrpn_float64 quat[4];
        } vrpn_TRACKERCB;
```

Here the variable 'sensor' with value 0 and 1 is for left and right eye respectively, and unified view called cyclops gaze which is accessed by sensor value 2 . 'msg_time' gives the time stamp of the generated data. Variable 'pos' is the position in 3D-pose x,y,z for respective axis and 'quat' is the gaze direction in quaternion terms. The quaternion can be transformed to euler angles using an inbuild matlab function called

```
eul = quat2eul(quat)
```

In order to test the functioning of the headtracking-eyetracking setup, a VRPN test application is included. The application provides the gaze vector of the user as show in Figure 3.11. The UI shows three lines,these are the sensor values mentioned before 0 for left 1 for right.

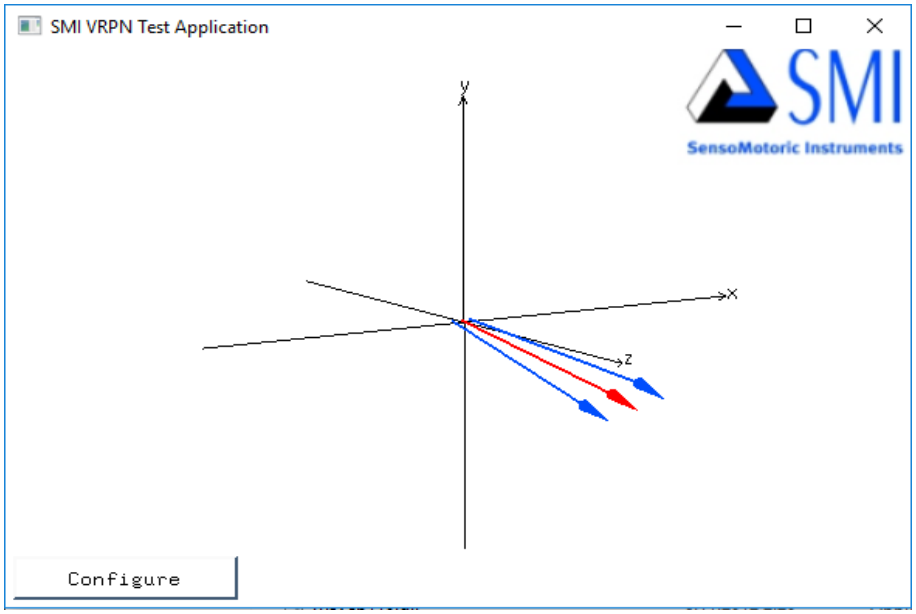


Figure 3.11 *VRPN test application with gaze direction*

4. METHODOLOGY AND IMPLEMENTATIONS

In the first Section, of this chapter the setup of cameras is explained. The second Section describes software and hardware setup, and their integration.

4.1 Environment Setup

In this project, the area of interest for motion tracking is approximately 2 x 2 meters. This can be effectively captured by a Motion capture system consisting of 8 Flex 3 cameras. In order to connect these 8 cameras to the computer, two Optihub 2 systems are used. Each optihub can connect upto six cameras but four cameras are connected to each hub due to physical constraints and better cable management. The cameras are powered via Optihub 2 system.

The position of the cameras are chosen as shown in Figure 4.1. The cameras are placed in order to provide maximum coverage from any angle inside the capture space as high marker visibility leads to higher accuracy in tracking.

The Turtlebot is fully charged as lack of power might lead to malfunctioning of the actuators. The reflective markers are setup on the topmost frame of the Turtlebot around the Laser scanner. The asymmetric shape of the markers are as shown in Figure 4.2. This helps in defining a rigid body which can be tracked efficiently via software as choosing a symmetric shape leads to misleading data.

4.2 Hardware and Software setup

The cameras are calibrated in Motive and the origin of the capture volume is set with Y-plane as the floor (Y=0) . Its important to choose the origin as shown in Figure 4.1 so as to get maximum working area for the TB. After this, the rigid bodies i.e. TB and ETG Head tracking gear, are defined in Motive, the camera settings can be adjusted at this time to suit the external light in the room. The rigid body id of both system are noted for future use.

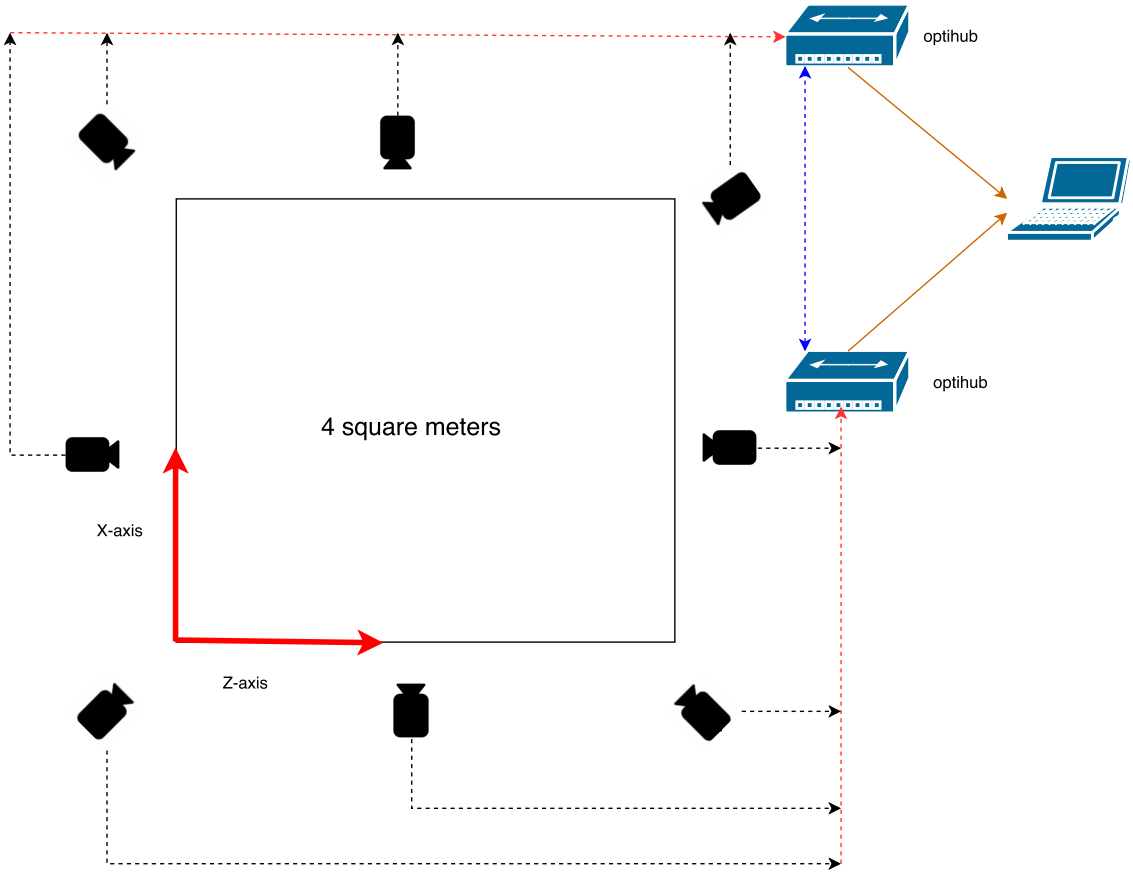


Figure 4.1 *Camera setup for motive with coordinate axis present in the 3D virtual space*

Streaming of data is started in local host and VRPN network , this can be checked in the VRPN server program . Eye Tracking glasses are connected to the smart phone via USB cable. Once the glasses are calibrated, its connected to the same network to which the user PC is connected. The IP address of the connection is shown on the user interface of the software which is entered into VRPN application. Also the information of the Motive stream is entered into the application as shown

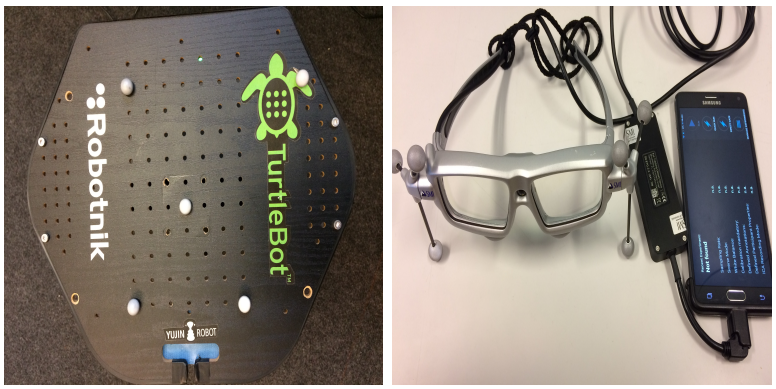


Figure 4.2 *Marker setup on top on turtlebot(Left) ETG head tracking setup with PDA for connecting with the network(Right)*

in Figure 4.3. Once connected we get green lines on for both systems, now we can setup Matlab and ROS .

Once the TurtleBot (TB) is powered up, the user can connect to the system via PuTTY: SSH client available for windows, using the IP address of the TB .

Both systems are pinged to avoid any communication errors. Once the TB is connected, in order to connect Matlab, Turtlebot software is started with the help of minimal.launch command as shown below

```
roslaunch turtlebot_bringup minimal.launch
```

Matlab is used to initiate communication between each other using the command .

```
ipaddress = '192.168.2.59';
rosinit(ipaddress)
```

Once Matlab successfully connects with TB, ,its time to setup data streaming between Motive and Matlab using Natnet library which is accomplished with the commands

```
HostIP = char('127.0.0.1')
theClient.Initialize(HostIP, HostIP)
RigidBody_ID=1
frameOfData = theClient.GetLastFrameOfData()
rigidBodyData = frameOfData.RigidBodies(RigidBody_ID)
Coord = [ rigidBodyData.x, rigidBodyData.y, rigidBodyData.z ]
```

The RigidBody_ID is set to the TB rigid body ID noted down before and the data is pulled from the local network and X, Y, Z coordinate data is saved to a variable.

The program written in cpp is executed which writes the quaternion data to the file, successful execution of the program can be noticed in VRPN network via the green line in merge data shown in Figure 4.3

This file is accessed by MatLab which copies the data to a variable in MatLab. We take the quaternion value and solve for Euler angle with equation three. After calculating euler angle for both gaze and head tracking using Equation (4.2). As shown in Figure 4.4, we can calculate the effective rotation by adding the two values. We can find the point at which the gaze vector crosses the Y-plane as follows, the calculations shown below is for Global coordinate system with Z-up.

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{4.1}$$

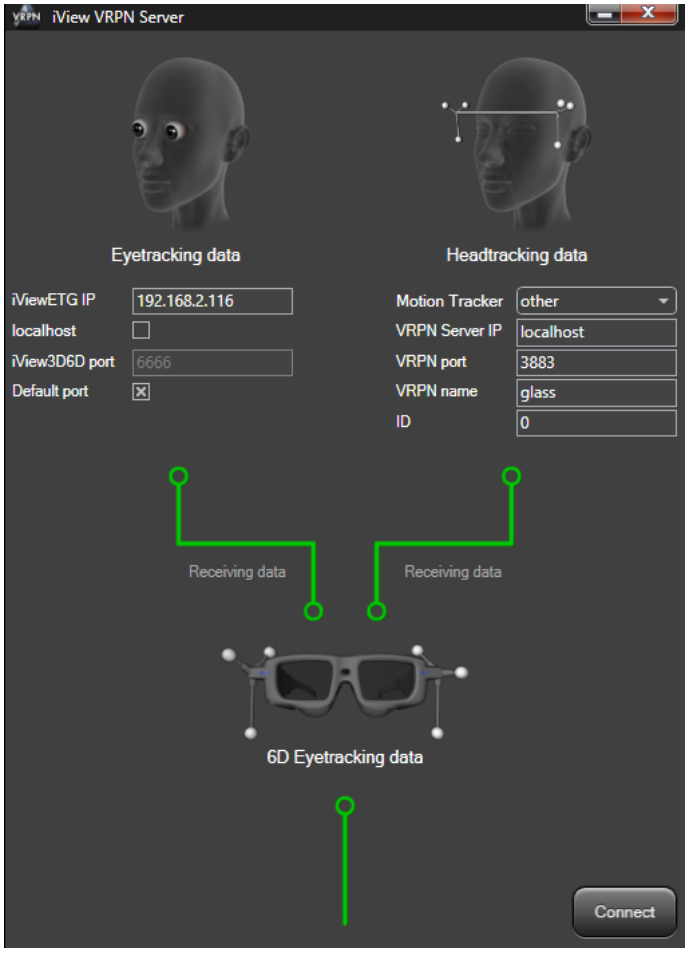
!hb 

Figure 4.3 *Successful data streaming in VRPN server application after connecting ETG and Motive*

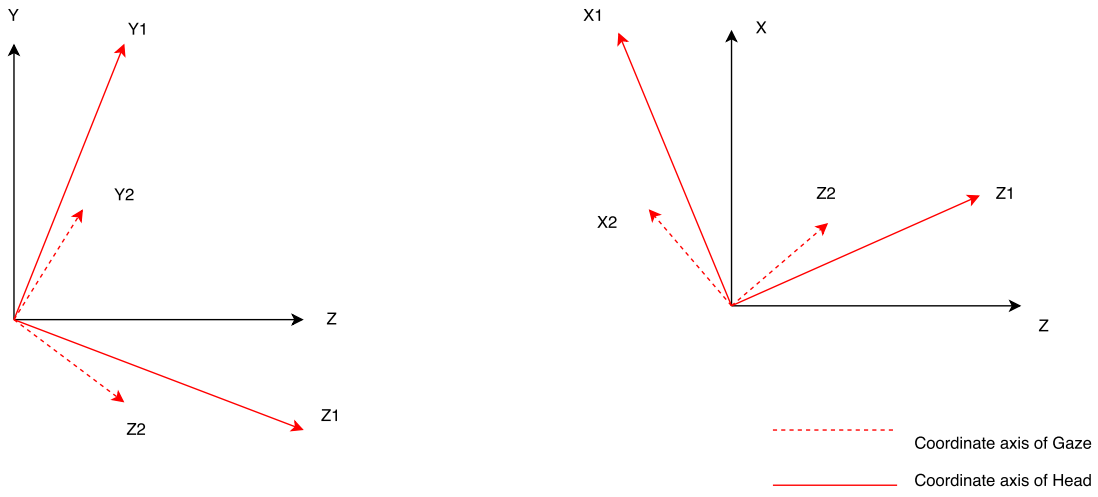


Figure 4.4 *Angle of rotation after taking into account gaze and head rotation*

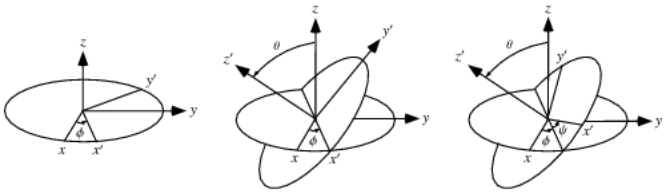


Figure 4.5 *Euler angle rotation in X-convention*

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0q_1+q_2q_3)}{1-2(q_1^2+q_2^2)} \\ \arcsin(2(q_0q_2 - q_3q_1)) \\ \arctan \frac{2(q_0q_3+q_2q_1)}{1-2(q_3^2+q_2^2)} \end{bmatrix} \tag{4.2}$$

Figure 4.5 show rotation across z-axis, x-axis and the former z-axis, which can be given by the rotation matrix[21]

$$A = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} C = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Head coordinates X', Y', Z' in global coordinate system X, Y, Z is given by dX=X-X', dY=Y-Y' , and dZ=Z-Z'. Hence using X-convention, i.e. rotation across A followed by B and C , point (dX, dY, dZ) can be written as

$$\begin{bmatrix} dX \\ dY \\ dZ \end{bmatrix} = ABC \begin{bmatrix} x \\ y \\ z \end{bmatrix} ; \rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} + ABC \begin{bmatrix} x \\ y \\ z \end{bmatrix} ; \tag{4.4}$$

Now gaze line in head coordinates can be presented as

$$x^{"} - t * n = 0; x^{"} = t * n; \tag{4.5}$$

where x" = (x, y, z);

As the gaze line passes through the point x = y = z = 0, i.e. the origin of the head coordinates. An unit vector 'n', rotating through an angle of 'alpha' in x direction and 'beta' in z direction i.e. horizontal and vertical movement of eye respectively,

can be represented as

$$\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.6}$$

Substituting Equation 4.6 in Equation 4.5, we get

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = t * \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

Hence from Equation 4.4, we get

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} + t * ABC \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.8}$$

Now the intersection of gaze line with any plane in global coordinates can be solved. For example, gaze line passing through Y-plane in global coordinates i.e. Y' = 0. We can find the value of t from equation 4.8 and solve for X' and Z', giving us (X', 0, Z'). Figure 4.6 shows a simple representation of gaze vector's intersection with Y-plane.

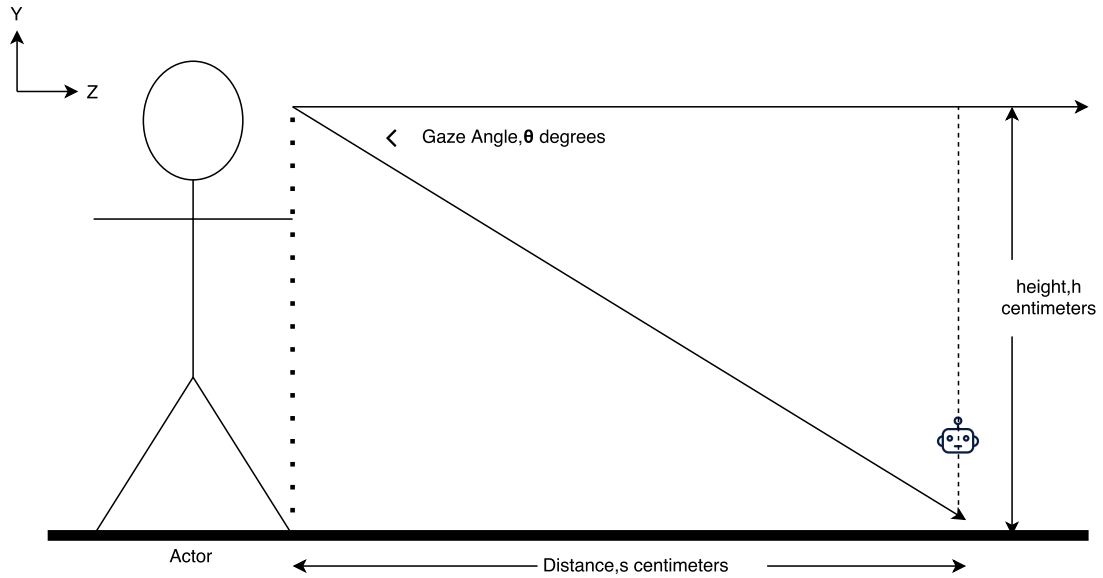


Figure 4.6 *Simple representation of gaze angle and gaze line*

Once the Final Coordinate is calculate using Vector gaze direction, difference between the final coordinate and initial coordinate is calculated, the distance and angle between the two coordinates are also calculated.

```
 %distance calculation
dist= sqrt((CoordOrg(1))^2+((CoordOrg(3))^2))
%angle calculation
theta=atan(abs(CoordOrg(3)/CoordOrg(1)))
     thetaindegree=theta*180/ pi
```

The TB is oriented by setting turnV value to 1 or -1 to rotate towards the final position. In order to move TB the 'turnV' is set to 0 and 'forwardV' is set to 1. The camera data is accessed to check whether the robot is moving in the correct direction which is accomplished by calculating the distance between final position and current position of the robot.

```
    velPub = rospublisher('/mobile_base/commands/velocity');
velMsg = rosmessage(velPub);

forwardV = 0; (value greater than 0 = robot movement)
turnV    =0;  (value greater than 0 = robot turn)
    velMsg.Linear.X = forwardV;
    velMsg.Angular.Z = turnV;

send(velPub,velMsg);
```

The Speed of the TB is kept minimum so that it can cover shorter distance while the processing is done in real time, increase in speed makes the TB to move past the final position if the processing times mismatch. The complete setup is shown in figure 4.7, and the algorithm of the control system is shown in figure 4.8

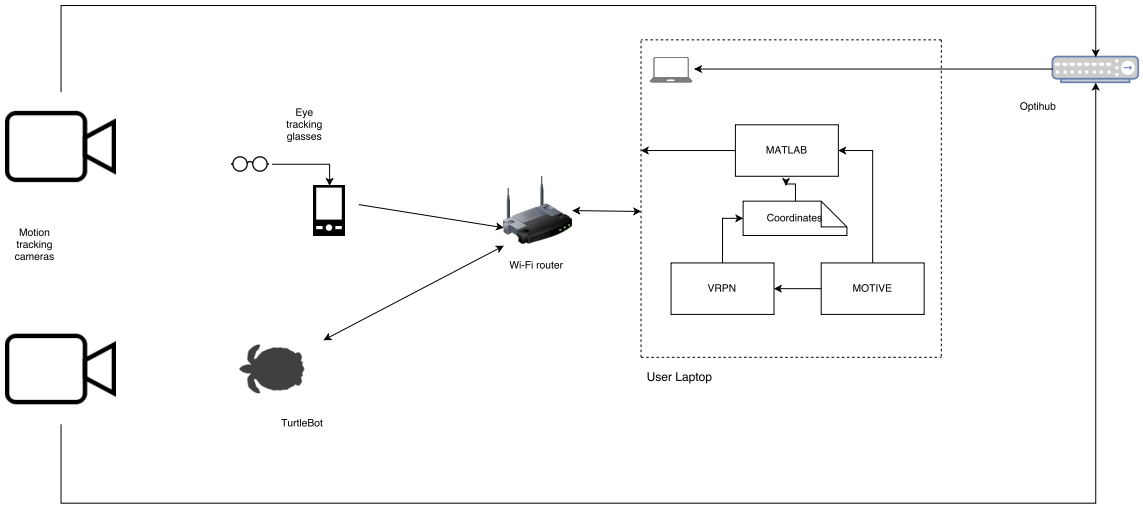


Figure 4.7 *Overview of hardware and software setup for TB, ETG and Motion capture system*

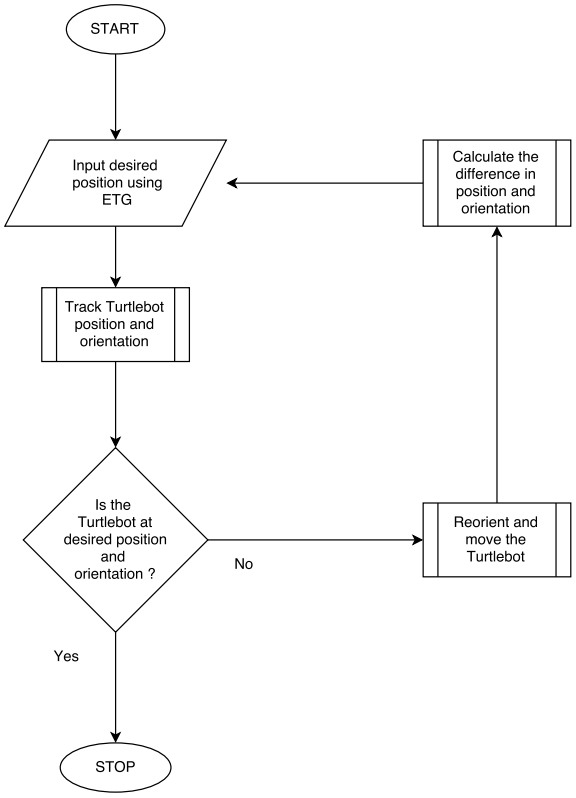


Figure 4.8 *Control system used for motion tracking of TB*

5. RESULTS

This section describes the results of trials conducted to calculate the error in TB reaching the end point and also the error in ETG gaze calculation.

Before discussing the result, as mentioned before TB movements are inaccurate as we are not using the odometer present in the robot and solely depending on the motion tracking cameras, this helps in designing an independent system with less communication but the downside of this system is the increase in error. When the robot is oriented towards the direction of the final position, the minimum angular movement of the robot should be taken into consideration as this affects the final position of the robot as small difference in the angle can drastically increase the error when the robot travels for longer distances. The availability of motion tracking helps in reducing this error and the trials are carried out to measure the amount of error in the system .

Figure 5.1 shows the test environment used to check the reliability of Camera Tracking of TB. Points are chosen such that the Robot gets to move and orient itself at different angles with respect to its initial position, thus calculating error for all possible conditions. For example, when the robot is at A and is commanded to move to E, the orientation can overshoot or undershoot resulting in a small error in the final direction of motion. This will change the end position by a small order. Similarly, when the robot is moving from point B to point E, small reorientation required might lead to an error.

Table 5.1 shows the error percentage obtained from the trials. There is an error of 5% along X and Z axis. This might be avoided by tracking the odometer and decreasing the orientation speed for precise rotation.

The error in gaze commanding is measured by choosing a fixed point and then using ETG to point it out.The data is recorded and the gaze point is calculated. Due to inaccurate reading in Z axis, equation 7, 8, and 9 is used. The results are tabulated in Table 5.2.

The amount of error is higher due to inaccuracy and taking only a data point instead

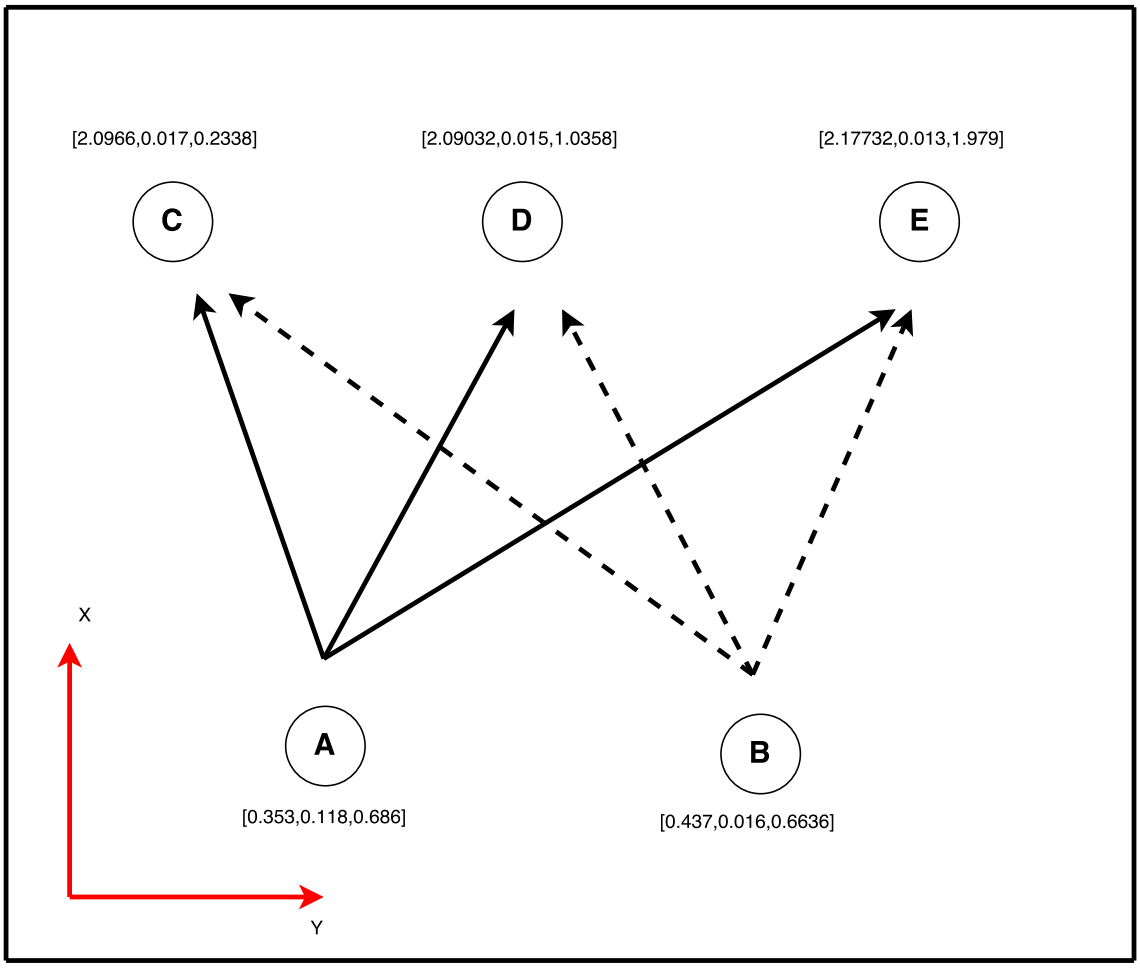


Figure 5.1 *Test Environment setup to check the reliability of Camera tracking .*

of a set of data for gaze calculation. Table 5.3 shows the measurements taken when using gaze to command TB. The error increases because of the cumulative effect of error because of ETG recording and TB movement.

Table 5.1 *Error percentage when TurtleBot is Motion Tracked from Initial Point to Predetermined Point in Test Area*

Trial No	Expected Final Position [X,Z] in mm	Final Position Reached from A=[353, 686] mm	Percentage Error X axis	Z axis
1	C =[2096, 233]	[1995, 210]	5.09	11.33
2	C =[2096, 233]	[2010, 238]	4.30	1.76
3	C =[2096, 233]	[1983, 279]	5.72	16
4	D=[2090, 1035]	[2008, 1141]	4.06	9.22
5	D=[2090, 1035]	[2026, 1156]	3.17	10.39
6	D=[2090, 1035]	[2048, 1168]	2.06	11.31
7	E=[2177, 1979]	[2082, 2067]	4.57	4.25
8	E=[2177, 1979]	[2059, 2039]	5.57	2.94
9	E=[2177, 1979]	[2074, 2084]	4.9	5.03

Trial No	Expected Final Position in [X,Z] in mm	Final Position Reached from B=[437, 1663]mm	Percentage Error X axis	Z axis
1	C=[2096, 233]	[1965, 279]	6.67	16
2	C=[2096, 233]	[1969, 257]	6.46	9.02
3	C=[2096, 233]	[2026, 248]	3.45	5.72
4	D=[2090, 1035]	[2015, 1153]	3.73	10.2
5	D=[2090, 1035]	[2031, 1147]	2.91	9.71
6	D=[2090, 1035]	[1991, 1127]	4.98	8.14
7	E=[2177, 1979]	[2050, 2046]	6.20	3.27
8	E=[2177, 1979]	[2108, 2063]	3.28	4.07
9	E=[2177, 1979]	[2118, 2062]	2.79	4.02

Table 5.2 *Percentage error between Perceived and Recorded Position*

Trial No	Position of Head Tracking Device(mm)	Orientation in Degrees along X axis	Expected Position(EP) in Z-axis (mm)	Calculated Position(CP) in Z-axis (mm)	Percentage Error %
1	2131,1174,-0109	32.0548	1.979	1766	10.76
2	2127,1186,-0072	32.3808	1.979	1797	9.15
3	2149,1169,-0136	31.5772	1.979	1766	10.76
4	2133,1178,-0180	31.2489	1.979	1762	10.96
5	2144,1172,-0162	30.8218	1979	1802	8.93
6	2136,1167,-0139	32.3966	1979	1699	14.11
7	2141,1186,-0160	31.1207	1979	1804	8.81
8	2124,1151,-0155	31.5536	1979	1720	13.06
9	2129,1166,-0155	31.2851	1979	1764	10.84
10	2135,1170,-0190	29.4440	1979	1882	4.85

Table 5.3 *Percentage error calculation of gaze commanding*

Trial No	Expected Position [X,Z] mm	Final Position [X,Z] mm	Percentage Error %
1	[1801,1829]	[1550,1545]	[16.19,18.38]
2	[1801,1829]	[1575,1562]	[14.34,17.09]
3	[1801,1829]	[1584,1554]	[13.69,17.69]
4	[1801,1829]	[1645,1574]	[9.48,16.20]
5	[1801,1829]	[1579,1545]	[14.05,18.38]

6. SUMMARY

The goal of this Master Thesis was to use Eye tracking glasses and use its feature to command a Turtlebot whose position is tracked using Motion tracking system. In order to achieve this goal, the working of hardware components and their respective software packages were studied and a simplified method to facilitate communication was created. This was achieved using MATLAB software and the Robot system toolbox provided in it, which is used to communicate with the ROS system present in the Turtlebot. Matlab also supports NatNetSDK which is a set of client / server library used to access the data from motion tracking system. VRPN is used to establish connection between ETG and motion tracking system to record gaze tracking and calculate the required final position pointed by the user.

This thesis has focused on developing communication between ETG,TB and Motion capture system.It also focused on commanding a TurtleBot using Eye tracking glasses. The results of the project show that establishment of communication can be carried out via Matlab using Robotics system toolbox and NatNet and iView Software development kit with some error in reaching the final position.

The future of the project lies in increasing the quadrants on which the robot operates which is restricted to one in this project. There is scope for extending the number and type of robots working simultaneously while communicating with each other as the program doesn't depend on the sensors reading of the robot.

Last but not least, the result of this project will act as a stepping stone to projects involving motion capture systems. In conclusion, motion capture system help in giving a sense of direction for robots which are otherwise need to depend on their internal mapping .

BIBLIOGRAPHY

[1] Microsoft, https://support.microsoft.com/en-us/help/12647/hololens-use-your-voice-with-hololens, *Use your voice with HoloLens*, revision: 24 ed., April 2017.

[2] L. group., "Lego mindstorm," 2017.

[3] A. Myers, "Stanford researchers adapt a diy robotics kit to give stem students tools to automate biology experiments," March 2017. http://news.stanford.edu/2017/03/21/adapting-diy-robot-kit-fill-test-tubes/.

[4] robotshop, "Arduino robot kits," 2017.

[5] robotshop, "Arduino programming language," 2017.

[6] S. Kang, J. Paik, A. Koschan, B. Abidi, and M. A. Abidi, "Real-time video tracking using ptz cameras," *Proc. SPIE 5132, Sixth International Conference on Quality Control by Artificial Vision,*, vol. 5132, p. 9, May 2003.

[7] A. Sharma, MukeshAgarwal, A. Sharma, and PankhuriDhuria, "Motion capture process, techniques and applications," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 1, pp. 251–257, April 2013. motion capture methods.

[8] Optitrack, "Optitrack documentation wiki," November 2016.

[9] H. Chennamma, "A survey on eye-gaze tracking techniques," *IJCSE*, vol. 4, p. 9, oct-nov 2013. https://arxiv.org/ftp/arxiv/papers/1312/1312.6410.pdf.

[10] R. G. LUPU and F. UNGUREANU, "A survey of eye tracking methods and applications," *IJCSE*, 2013.

[11] A. T. Duchowski, "A breadth-first survey of eye tracking applications," *Behavior Research Methods, Instruments, and Computers*, 2002.

[12] C. robotics, *Turtlebot 2 Personal Robot Usermanual*. Clearpath robotics, https://www.generationrobots.com/media/TurtleBot-2-UserManual.pdf, 1 ed., 2013.

[13] Z. M. S.-R. Neerendra Kumar, Zoltn Vmossy, "Robot obstacle avoidance using bumper event," *IEEE International Symposium on Applied Computational Intelligence and Informatic*, May 2016. bumper event Tb , http://ieeexplore.ieee.org/document/7507426/.

[14] MATLAB, "Robotics system toolbox examples," 2017.

[15] "comparision of hardware manufactured by optitrack." accessed June,2017.

[16] "Flex 3 camera specifications." Accessed June 2017.

[17] OptiTrack, http://optitrack.com/products/natnet-sdk/, *NatNet API Users Guide*, version 2.10.0 ed., May 2016. checked 10 july 2017.

[18] vrpn, "Virtual reality peripheral network - official repo," 2017. accessed June,2017.

[19] SMI, *iViewNG SDK*. SMI, v2.7.1 ed., JULY 2016. Supplied with the device , not available online.

[20] SMI, *SMI 3D Eye Tracking and 6D Head Tracking User Guide*. SMI, version 1.0.2 ed., March 2014. Supplied with the Hardware.

[21] W. MathWorld, "Euler angles,"

Appendices

Program 1 *VRPN server CPP program*

```
//cpp program to get data from ETG
#include "stdafx.h"
#include "vrpn_Shared.h"
#include "vrpn_Tracker.h"
#include <iostream>
#include <fstream>

using namespace std;
int i=0;



 void VRPN_CALLBACK handle_tracker(void *userdata, const
    vrpn_TRACKERCB t)
{
     if (t.sensor == 2)
     {

            ofstream myfile;
            myfile.open("coordinates.dat");
            myfile << t.quat[0]    << " ";
            myfile << t.quat[1]   << " ";
            myfile << t.quat[2]   << " ";
            myfile << t.quat[3]   << " ";
            myfile << t.quat[4]   << endl;
            myfile.close();


     }
     }



int main(int argc, char* argv[])
{
    // setup VRPN

    vrpn_Tracker_Remote* vrpnTracker = new
```

```
        vrpn_Tracker_Remote("SmiETG@localhost:4444");
    vrpnTracker->register_change_handler(0, handle_tracker);
    const unsigned int WAIT_PER_ITERATION = 1500; //
        milliseconds
    while (1)
    {
        vrpnTracker->mainloop();
        vrpn_SleepMsecs(WAIT_PER_ITERATION);
    }
    return 0;
}
```

appendix:cppcode

```
%%MATLAB code to connnect and control TB
%initiate connection to ROS

clear all
clc
addpath('C:\Users\shetty\Dropbox\Thesis\Matlab Files\Camonly
    ')
ipaddress = '192.168.2.59';
rosshutdown
rosinit(ipaddress)
rostopic list
velPub = rospublisher('/mobile_base/commands/velocity');
velMsg = rosmessage(velPub);
%wait for beep= confirmation of connection (pub and rec)

soundpub = rospublisher('/mobile_base/commands/sound','
    kobuki_msgs/Sound');
soundmsg= rosmessage('kobuki_msgs/Sound');
soundmsg.Value = 2;
send (soundpub,soundmsg);
%%
%get final coordinates from Glasses savce is finCoord
Aflag=0;
Dflag=0;
TurnBck=0;
yaw1=0;
```

```
25
26
27
28
29
30 %finCoord1=[1.030,0,1.974]
31 finCoord1 = gaze();
32 finCoord=finCoord1+ [finCoord1(1) ,0,finCoord1(3)];
33
34 %prompt=('1.Press Enter to initiate\n')
35 %userIP = input(prompt)
36
37 [dis,b,yaw,p]=CamFunc(finCoord);
38 b
39
40
41 %Angle
42           while(Aflag~=1)
43             %turning
44              pause(1)
45           if (abs(b)<3)
46                    Aflag=1;
47                    prompt=('else1\n')
48                    break
49           elseif(yaw1<abs(b)) && (b<0)  %and its negative
50               %if p==0
51                turninput(+.4);
52                 pause(2)
53                [dis,extra,yaw,p]=CamFunc(finCoord);
54               yaw1=yaw
55               %end
56
57           elseif (abs(yaw1)<abs(b)) && (b>0) %and its positive
58                 turninput(-.4);
59                 pause(2)
60                [dis,extra,yaw,p]=CamFunc(finCoord);
61               b
62               yaw1=yaw
63
```

```
64
65          else
66
67              Aflag=1;
68          end
69      end
70
71
72
73
74
75  %Distance
76      while(Dflag~=1)
77
78                      %movement
79
80                  [dis,b,yaw,p]=CamFunc(finCoord);
81                  if(dis<=0.15)
82                          Dflag=1; %return
83              TurnBck=1;
84              break
85              end
86          velocityinput(.2)
87
88      end
89
90
91
92
93  theClient.Uninitialize
```

```
1  function turninput(turnV)
2
3      %publish to ROS
4      velPub = rospublisher('/mobile_base/commands/velocity');
5      velMsg = rosmessage(velPub);
6
7
8      %set minimum turn per input
9
```

```
10
11      forwardV = 0;
12
13
14      %send message
15
16          if isempty(velMsg)
17              velMsg = rosmessage(velPub);
18          end
19
20              velMsg.Linear.X = forwardV;
21              velMsg.Angular.Z = turnV;
22
23
24              send(velPub,velMsg);
25
26
27
28  end
```

```
1   function velocityinput(forwardV)
2           %publish to ROS
3       velPub = rospublisher('/mobile_base/commands/velocity');
4       velMsg = rosmessage(velPub);
5
6       %set minimum speed per input
7
8
9       %forwardV = 0.5;
10          turnV=0;
11
12      %send message
13
14
15      if isempty(velMsg)
16              velMsg = rosmessage(velPub);
17      end
18
19              velMsg.Linear.X = forwardV;
20              velMsg.Angular.Z = turnV;
```

```
21
22                 send(velPub,velMsg);
23  pause(1)
24 end
```

```
1  function [dis,y,yaw,pitch]=CamFunc(finCoord)
2
3          % MOTIVE MUST BE STREAMING SOME DATA - RIGID BODY
               FOR EX , ELSE YOU GET
4  % ERROR MESSAGE OR MATLAB CRASHES
5
6  % thanks to NatNet (Natural Point Motive/Arena for
       Optitrack) location data simple sample in Matlab by Or
       Hirshfeld
7  %local loop back for server
8  addpath('C:\NatNetSDK\Samples\Matlab')%NatNetSDK folder
       location
9  %Add NatNet .NET assembly so that Matlab can access its
       methods, delegates, etc.
10 % Note : The NatNetML.DLL assembly depends on NatNet.dll, so
        make sure they
11 % are both in the same folder and/or path if you move them.
12 dllPath = fullfile('c:','NatNetSDK','lib','x64','NatNetML.
       dll');
13 assemblyInfo = NET.addAssembly(dllPath);
14 theClient = NatNetML.NatNetClientML(0) ;%Motive settings :
       Input = iConnectionType: 0 = Multicast, 1 = Unicast
15 %Define Final Position
16 %connect to server
17 HostIP = char('127.0.0.1');
18 theClient.Initialize(HostIP, HostIP) ;
19 RigidBody_ID=1;
20 %Flg = returnCode: 0 = Success % If connection fail , return
        0
21 frameOfData = theClient.GetLastFrameOfData();
22 rigidBodyData = frameOfData.RigidBodies(RigidBody_ID);
23 presCoord = [ rigidBodyData.x, rigidBodyData.y,
       rigidBodyData.z ];
24 % angle
25 q = quaternion( rigidBodyData.qx, rigidBodyData.qy,
```

```
    rigidBodyData.qz, rigidBodyData.qw ); % extrnal file
    quaternion.m
qRot = quaternion( 0, 0, 0, 1);    % rotate pitch 180 to
    avoid 180/-180 flip for nicer graphing
q = mtimes(q, qRot);
angles = EulerAngles(q,'zyx');
yaw = angles(2) * 180.0 / pi;
pitch = -angles(1) * 180.0 / pi;  % must invert due to 180
    flip above
roll = -angles(3) * 180.0 / pi ; % must invert due to 180
    flip above
```

```
%calculation
presCoord

u=finCoord-presCoord;
dis=sqrt((u(3))^2+((u(1))^2));

%save initial coordinate in another place , as the initial
    coordinate changes with rotation

     %ang=(atan(u(3)/u(1)))*180/pi
     ang=atan2d(u(3),u(1));
     if(abs(ang)>3)  % it was 3 before
     y=ang;
     else
     y=0;
     end
theClient.Uninitialize()

end
```

```
function [gazecoord] = gaze()
addpath('C:\NatNetSDK\Samples\Matlab')%NatNetSDK folder
    location
%Add NatNet .NET assembly so that Matlab can access its
```

```
    methods, delegates, etc.
% Note : The NatNetML.DLL assembly depends on NatNet.dll, so make sure they
% are both in the same folder and/or path if you move them.
dllPath = fullfile('c:','NatNetSDK','lib','x64','NatNetML.dll');
assemblyInfo = NET.addAssembly(dllPath);
theClient = NatNetML.NatNetClientML(0) ;%Motive settings : Input = iConnectionType: 0 = Multicast, 1 = Unicast
%Define Final Position
%connect to server
HostIP = char('127.0.0.1');
theClient.Initialize(HostIP, HostIP) ;
RigidBody_ID=2;
%Flg = returnCode: 0 = Success % If connection fail , return 0
Data(1,1)=0;
addpath('C:\Users\shetty\Documents\Visual Studio 2015\Projects\ConsoleApplication1\ConsoleApplication1')
%quaternion value from the file coordinates.dat
while Data(1,1) == 0
    pause(1);
Data=importdata('coordinates.dat');
if Data(1,1) == 1
    quat=[Data(2,1),Data(2,2),Data(2,3),Data(2,4)];
frameOfData = theClient.GetLastFrameOfData();
rigidBodyData = frameOfData.RigidBodies(RigidBody_ID);
presCoord = [ rigidBodyData.x, rigidBodyData.y, rigidBodyData.z ];
% angle
q = quaternion( rigidBodyData.qx, rigidBodyData.qy, rigidBodyData.qz, rigidBodyData.qw ); % extrnal file quaternion.m


qRot = quaternion( 0, 0, 0, 1);    % rotate pitch 180 to avoid 180/-180 flip for nicer graphing
q = mtimes(q, qRot);
angles = EulerAngles(q,'zyx');
```

```
33 A=rad2deg(angles);
34 % body defined while looking towards X direction ( computers
       )
35 %to left its positive
36 %to right its negative
37 %to down its negative
38
39 %yaw = angles(2) * 180.0 / pi;
40 %pitch = -angles(1) * 180.0 / pi;  % must invert due to 180
       flip above
41 %roll = -angles(3) * 180.0 / pi ; % must invert due to 180
       flip above
42
43 end
44 end
45
46 %For glass
47 %To the right its positive
48 %to the left its negative
49 eulZYX = quat2eul(quat);
50 D = rad2deg(eulZYX);
51 d1=180-D(1);% depending on the axis
52 d2=(-1*D(2));
53
54 RollZ=d1+A(3);
55 RollY=d2+abs(A(2));
56
57
58
59 z=tand(RollZ);  %%tan(x)=height/base
60 dis=abs(presCoord(2)/z);
61 CalX = presCoord(1) + dis * (cosd(RollY));
62
63 CalZ = presCoord(3) + dis * (cosd(RollY));
64
65
66
67 gazecoord=[CalX,0,CalZ]
68 theClient.Uninitialize()
```

```
69 end
```